Práctica 7. TIPOS DE DATOS ESTRUCTURADOS

Arrays

 2 sesiones

Semanas: 18 y 25 de Noviembre 

OBJETIVOS:

- ✓ Conocer la importancia que tienen los arrays como estructuras de datos en el ámbito de la programación.
- ✓ Conocer los distintos tipos de arrays.
- ✓ Conocer las operaciones posibles a realizar sobre un array.
- ✓ Aprender alguno de los métodos de búsqueda y ordenación de estructuras de datos de tipo array.

Veamos antes de trabajar con ellas algunas definiciones con las que nos debemos familiarizar:

ARRAY.- estructura de datos constituida por un número fijo de elementos, todos ellos del mismo tipo y ubicados en direcciones de memoria físicamente contiguas.

ELEMENTOS DEL ARRAY.- también denominado componente, es cada uno de los datos que forman parte integrante de la array.

NOMBRE DE UNA ARRAY .- identificador utilizado para referenciar el array y los elementos que lo forman.

TIPO DE DATO DE UN ARRAY .- determina el tipo de dato que es común a todos y cada uno de los elementos o componentes que forman dicho array (numérico, carácter, etc.).

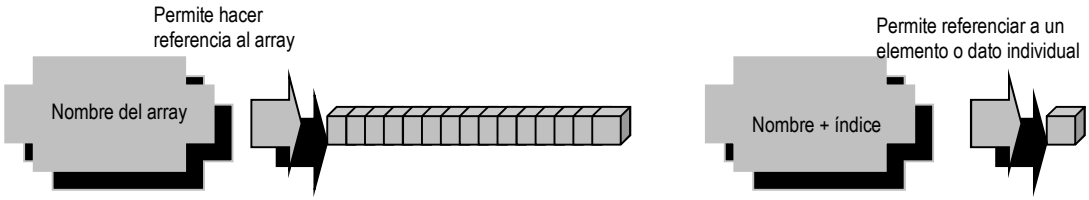
ÍNDICE .- normalmente es un valor numérico entero y positivo a través del cual podemos acceder directa e individualmente a los distintos elementos de un array, pues marca la situación relativa de cada elemento o componente dentro del mismo. En general, se puede utilizar como índice de un array un tipo ordinal (enteros y caracteres). Los tipos ordinales son aquellos que dado un elemento del dominio de ese tipo se encuentra definido su sucesor y/o predecesor.

TAMAÑO DE UNA ARRAY .- el tamaño o longitud de un array viene determinado por el número máximo de elementos que la forman, siendo el tamaño mínimo un elemento, y el tamaño máximo n elementos ($n > 0$).

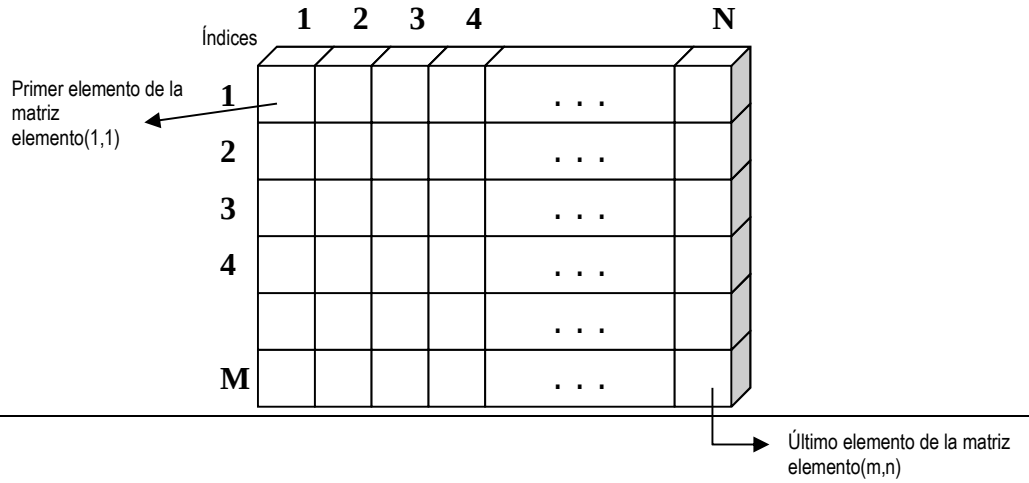
ACCESO A LOS ELEMENTOS O COMPONENTES DE UN ARRAY (ÍNDICE) .- los elementos o componentes de un array tratados individualmente son auténticos datos básicos del tipo correspondiente definidos para dicho array, reciben el mismo trato que cualquier otra variable de ese tipo, y tienen una denominación propia que les distingue del resto de elementos o componentes que constituyen dicha estructura de datos. Para acceder o referenciar un elemento en particular es suficiente con indicar el nombre del array seguido del índice (posición relativa que ocupa dicho elemento dentro del array) correspondiente a dicho elemento entre corchetes: nombre_array[indice].

DIMENSIÓN DEL ARRAY.- viene determinada por el número de índices que necesitamos para acceder a cualquiera de los elementos que lo forman.

Existen arrays de una dimensión (unidimensionales).



Y arrays de más de una dimensión. Como ejemplo de array n-dimensional podemos ver un array de 2 dimensiones. En este caso particular de 2 dimensiones se le denomina **matriz** o array bidimensional.



Ejercicio Resuelto 1. **Realiza un programa en C que lea 5 números reales y calcule su media, varianza y desviación típica. Media= ($\sum x_i$) / n Var = $\sum (x_i$ _ media)^2 Desv = var$^{(1/2)}$**

Para resolver este problema hay que almacenar los números en un array. La razón es la siguiente: cuando se calcula la media de una serie de números no es necesario almacenarlos, se pueden ir leyendo uno a uno y acumular su suma para luego calcular la media. Pero en este caso, sí es necesario guardar los números para poder calcular la correspondiente desviación respecto de la media de cada número.

```
#include <iostream>
#include <cmath>
using namespace std;

// declaración de la constante con el tamaño del vector de números reales
 const int TAM = 5;

// declaración de un nuevo tipo TVector
  typedef float TVector[TAM];

// cuerpo principal del algoritmo

int main()
{
   float   media, var, desv, suma;
   int     i;
   TVector vector;
   // inicializa la variable que acumula la suma de los cinco números
   suma = 0.0;

   // lee los cinco números y calcula su suma
   for (i=0; i < TAM; i++) {
      cout << "Introduce un numero: ";
      cin >> vector[i];
      suma = suma + vector[i];
   }

   // calcula y escribe la media
   media = suma / TAM;
   cout << "La media es " << media << endl;

   // calcula y escribe la varianza
   suma = 0;
   for (i=0; i < TAM; i++) {
      suma = suma + pow(vector[i] - media,2);
   }
   var = suma;
   cout << "la varianza es " << var << endl;

   // calcula y escribe la desviación típica
   desv = pow(var,1/2.0);
   cout << "la desviacion tipica es " << desv << endl;
}
```

Ejercicio Resuelto 2. **Realiza un programa en C en el que leamos un vector de enteros, obligando a que todos los enteros introducidos sean positivos. Luego ordenaremos el vector en modo ascendente y lo mostraremos ordenado. Tendremos un subprograma para leer el vector, otro para ordenarlo, y otro para imprimirlo.**

```
#include <iostream>
using namespace std;

// declaración de la constante con el tamaño del vector de números enteros
const int TAM = 10;

// declaración de un nuevo tipo TVector
typedef int TVector[TAM];

// Funciones cabecera
void leerVector(TVector vector);
void ordenaVector(TVector vector);
void imprimirVector(TVector vector);
;

// cuerpo principal del algoritmo
int main()
{
   TVector vector;
   leerVector(vector);
   ordenaVector(vector);
   imprimirVector(vector);
}

// Subprograma para leer el contentido del vector.
// Leeremos numeros enteros positivos
void leerVector(TVector vector)
{
    int i = 0;
    do {
       cout << "Introduzca el numero de la posición " << i << ": ";
       cin >> vector[i];
       if (vector[i] < 0)
          cout << "Error, debe introducir números positivos." << endl;
       else
          i++; // Pasamos a la siguiente posicion.
    } while (i < TAM);
}

// procedimiento que ordena el vector de forma ascendente.
void ordenaVector(TVector vector)
{
    int aux,i,j;
    for(i = 0; i < TAM-1; i++)
      for (j = (i+1); j<TAM ;j++)
       if(vector[j] < vector[i]) {
           aux = vector[i];
          vector[i] = vector[j];
          vector[j] = aux;
       }
}

// Imprime por pantalla los elementos del vector.
void imprimirVector(TVector vector)
{
    int i;

    cout << "Valores del vector: " << endl;
    for(i = 0; i < TAM; i++)
        cout << vector[i] << "  ";
    cout << endl;
}
```

Ejercicio Resuelto 3. **Realizar un programa en C con módulos que nos permitan leer los datos de un array bidimensional o matriz de 3 filas por 2 columnas y luego imprimir esta matriz por pantalla.**

```
#include <iostream>
using namespace std;

const int kFILAS = 3;
const int kCOLS  = 2;

typedef int TMatriz[kFILAS][kCOLS];

void leerMatriz(TMatriz matriz, int filas, int cols);
void escribirMatriz(TMatriz matriz, int filas, int cols);

int main()
{
    TMatriz matriz;

    leerMatriz(matriz, kFILAS, kCOLS);
    escribirMatriz(matriz, kFILAS, kCOLS);
}


// Procedimiento para leer los elementos de la matriz.
void leerMatriz(TMatriz matriz, int filas, int columnas)
{
   int i, j;

   for(i=0; i< filas; i++)
      for(j=0; j< columnas; j++) {
         cout << "Introduzca el elemento (" << i << "," << j << ") : ";
         cin >> matriz[i][j];
      }
}
// Procedimiento para imprimir la matriz.
void escribirMatriz(TMatriz matriz, int filas, int columnas)
{
   int i, j;

   for(i=0; i< filas; i++) {
      for(j=0; j< columnas; j++) {
         cout.width(5); // formatea salida rellenando con blancos según ancho indicado
         cout << matriz[i][j];
      }
      cout << endl;
   }
}
```

Ejercicio Resuelto 4. **Realiza un programa en C que dadas dos palabras, sustituya las letras minúsculas por mayúsculas, nos diga la longitud de cada palabra y nos indique cuál es la mayor en orden alfabético.**

```
#include <iostream>
using namespace std;

// declaración de la constante con el tamaño máximo de las palabras
const int TAM = 25;

// declaración de un nuevo tipo para las palabras
typedef char TPalabra[TAM];

// declaración de módulos
void Pasar_A_Mayusculas(char  cadena[]);

int main()
{
   TPalabra  palabra1, palabra2;
   int       compara;

   // leer las dos palabras introducidas por el usuario
   cout << "Introduce la primera palabra:";
   cin.getline(palabra1, TAM);
   cout << "Introduce la segunda palabra:";
   cin.getline(palabra2, TAM);

   // pasar a mayúsculas ambas palabras
   Pasar_A_Mayusculas(palabra1);
   Pasar_A_Mayusculas(palabra2);

   // mostrar la longitud de cada palabra
   cout << "la longitud de " << palabra1 << " es:" << strlen(palabra1) << endl;
   cout << "la longitud de " << palabra2 << " es:" << strlen(palabra2) << endl;

          // Comparar ambas palabras
   compara = strcmp(palabra1, palabra2);
   if (compara == 0)
      cout << "ambas palabras son iguales";
   else if (compara < 0)
      cout << palabra1 << " es menor que " << palabra2;
   else
      cout << palabra2 << " es menor que " << palabra1;

   cout << endl;
}

// Procedimiento que convierte una cadena de caracteres a mayúsculas
void Pasar_A_Mayusculas(char  cadena[])
{
   int  i;

   i = 0;
   while (cadena[i] != '\0') {
      cadena[i] = toupper(cadena[i]);
      i++;
   }
}
```

Ejercicio Resuelto 5. **Implementa un programa en C que solicite al usuario dos palabras y las compare según su orden alfabético. Para ello, primero se deberá validar que ambas palabras estén formadas exclusivamente por letras minúsculas ('a'..'z') o letras mayúsculas ('A'..'Z'). A continuación, se sustituirán en ambas palabras las letras mayúsculas por su correspondiente minúscula. Finalmente, se mostrarán por pantalla las 2 palabras y un mensaje indicando si son iguales, o cuál es la menor según el orden alfabético.**

```
#include <iostream>
using namespace std;

// Tamaño máximo de las cadenas
const int TAM = 20;

// Tipo de datos para las cadenas
typedef char TCadena[TAM];

bool esAlfabetica(TCadena);
void aMinusculas(TCadena);
void leerCadena(TCadena);

int main() {

      TCadena cad1, cad2;

      // Pide al usuario introducir dos cadenas
      leerCadena(cad1);
      leerCadena(cad2);

      // Convierte las cadenas a minúsculas
      aMinusculas(cad1);
      aMinusculas(cad2);

      // Muestra las cadenas en minúsculas
      cout << "Las dos palabras en minúsculas son:" << endl;
      cout << cad1 << endl << cad2 << endl;

      // Comprueba cuál es la menor cadena
      int result = strcmp(cad1,cad2);

      if(result<0) {
               cout << "La menor es " << cad1 << endl;
      } else if(result>0) {
               cout << "La menor es " << cad2 << endl;
      } else {
               cout << "Ambas cadenas con iguales" << endl;
      }
}

void leerCadena(TCadena cad) {
      do {
        cout << "Introduce una palabra: ";
        cin >> cad;

        if(!esAlfabetica(cad)) {
               cout << "Error, la palabra sólo debe contener letras" << endl;
        }
      } while(!esAlfabetica(cad));
}

bool esAlfabetica(TCadena cad) {
      bool alfabetica = true;
      int i=0;

      while(i<TAM && cad[i]!='\0') {
        if(!isalpha(cad[i])) {
               alfabetica = false;
               break;
        }
        i++;
      }

      return alfabetica;
}

void aMinusculas(TCadena cad) {
```

```
      int i=0;
      while(i<TAM && cad[i]!='\0') {
        cad[i] = tolower(cad[i]);
        i++;
      }
}
```

Ejemplo de ejecución:

Introduce una palabra: pizarra2
Error, la palabra sólo debe contener letras
Introduce una palabra: Pizarra
Introduce una palabra: boRRadOr
Las 2 palabras en minúsculas son:
pizarra
borrador
y la menor es: borrador

PARÁMETROS DE LA LÍNEA DE COMANDOS.

Hasta ahora los programas que se han realizado han tenido la misma estructura básica:

```
main( )
{
  . . .
}
```

Esta estructura se hace familiar ya que en definitiva es la sintaxis empleada para escribir funciones. Podemos completar nuestra función main de forma que podamos pasarle parámetros al programa. Para ello, se dispone de la siguiente estructura:

```
int main(int argc, char *argv[])
{
  . . .
  return n;
}
```

Programa que hace de interfaz entre el usuario y S.O.

Este es el formato de una función que devuelve un valor entero que puede ser recogido por el intérprete de comandos **Shell** y los parámetros de los paréntesis nos permiten pasar información desde el Shell a nuestro programa. En definitiva Unix dispone de un muy potente sistema de programación en su Shell, permitiéndonos realizar funciones complejas así como invocar programas y que éstos devuelvan un resultado en un valor entero, que por convenio será 0 si la ejecución ha sido correcta y otro valor entero para determinar errores (recuerda que este criterio es lo contrario del valor de verdadero y falso en C).

Desde la línea de comandos se inicia el programa especificando el nombre del archivo que contiene el programa compilado, y si deseamos pasar información a nuestra función main, deberemos escribir dicha información a continuación del nombre del fichero ejecutable. De la forma:

```
nombre-fichero-ejecutable parametro1 parametro2 . . . parametroN
```

Estos datos se podrán recoger en nuestro programa accediendo a la variable **argv**, que es un array de string (cadenas de caracteres). Y en **argc** se facilita un contador de cuántos parámetros se han empleado.

En argv[0] se almacena el nombre del programa. Por tanto, si se pasan N parámetros, habrá N+1 elementos en argv, entre argv[0] y argv[N] y argc tendrá asignado el valor N+1.

Un ejemplo sencillo del uso de los parámetros del `main`, lo vemos en el siguiente programa:

```
//prog.c
#include <iostream>
using namespace std;
// cuerpo principal del algoritmo
int main(int argc, char *argv[])
{
   int     contador;
   // recorrer los parámetros
   for (contador=0; contador < argc; contador++) {
      cout << "parámetro-" << contador <<": ";
      cout << argv[contador] << endl;
   }
}
```

Compilamos de la manera habitual:

```
g++ -o prog prog.c
```

Ejecutamos el programa:

```
./prog agua fuego tierra
```

El resultado:

```
parámetro-0: ./prog
parámetro-1: agua
parámetro-2: fuego
parámetro-3: tierra
```

Ejercicio Resuelto 6. **Implementa un programa en C que recoja los parámetros desde la línea de comandos del Shell y devuelva un resultado numérico de CERO cuando no hay parámetros repetidos, o en caso de existir repetidos, indicar cuántos lo están.**

```
#include <iostream>
#include <cstring>
using namespace std;

// funcion para buscar repetidos
/* busca el valor de lista de la posicion elementoBase hasta la posicion
indiceHasta.
devuelve un entero con la posición del duplicado o cero si no está
duplicado.*/
int buscar(int elementoBase, int indiceHasta,char *lista[]);

// cuerpo principal del algoritmo

int main(int argc, char *argv[])
{
   int     contador;
   int nuevaposicion;
   int elementosrepetidos;

   elementosrepetidos=0;
   for (contador=1; contador < argc; contador++) {
      nuevaposicion=buscar(contador,argc,argv);
      if (nuevaposicion>0) {
         elementosrepetidos++;
      }
   }
   return elementosrepetidos;
}

// funcion para buscar repetidos
/* tomando el elemento desde, busca en la lista hasta la posición hasta
Devuelve 0 si no encuentra repetido
O el número de la posición del elemento encontrado */
int buscar(int desde, int hasta, char *lista[])
{
  int i;
  int encontrado;  // se utiliza como indicador para salir
  encontrado=0;

  for (i=desde+1; i<hasta && encontrado==0 ; i++) {
    cout << "comparando.:" << lista[desde] <<"con:" <<lista[i] <<endl;
    if (strcmp(lista[desde],lista[i])==0) {
      encontrado=i;
    }
  }
return encontrado;
}
```

Para probar el programa en un script debes crear un fichero script.

Archivo que contiene una serie de instrucciones

Si el anterior programa se llama `p7_er6.c`

`g++ -o p7_er6 p7_er6.c` // creará el fichero ejecutable

el script p7_er6.sc contendrá al menos

```
#ejecuta el ejercicio resuelto 6 de la práctica 7
# recibe los mismos parámetros que le pasamos al script
./p7_er6 $@
# la orden echo muestra en pantalla
# el parámetro dólar interrogante es el resultado de la ejecución
echo resultado.: $?
echo ------------
```

Para poder ejecutar el script debes asignarle permisos de ejecución con:

```
chmod +x p7_er6.sc
```

Ejemplos de ejecución:

Caso1:

```
./p7_er6.sc agua tierra fuego
resultado.:0
------------
```

Caso2:

```
./p7_er6.sc agua tierra agua fuego
resultado.:1
------------
```

Caso3:

```
./p7_er6.sc a b a b c d e
resultado.:2
------------
```

Caso4:

```
./p7_er6.sc a b a a c d e
resultado.:2
------------
```

// la resolución de este error se propone como ejercicio.

Propuesto 1. **Implementar un programa en C que lea una matriz y calcule la multiplicación por su transpuesta. En primer lugar pedirá al usuario las dimensiones M x N de la matriz (filas x columnas), que como máximo podrá ser de 10x10.**

```
Introduce el número de filas: 2
Introduce el número de columnas: 3
```

Una vez hecho esto, se irán introduciendo uno a uno los elementos de la matriz.

```
Introduce el elemento (0,0): 2
Introduce el elemento (0,1): 3
Introduce el elemento (0,2): 3
Introduce el elemento (1,0): 1
Introduce el elemento (1,1): 4
Introduce el elemento (1,2): 2
```

2	3	3
1	4	2

A

2	1
3	4
3	2

A^T

Al terminar de introducir estos elementos, se calculará la matriz resultante de multiplicar la matriz introducida por su transpuesta, y se mostrará en pantalla.

```
22  20
20  21
```

<u>Ayuda</u>: Puedes utilizar como base el código del ejercicio resuelto 3.

Propuesto 2. **Implementa un programa en C que inicialmente visualice el siguiente menú:**

```
1)Introduce secuencia de números enteros
2)Visualiza secuencia en orden descendente
3)Buscar número
4)Salir
Opción (1/2/3/4)?
```

Dependiendo de la opción de menú elegida, se deben realizar las siguientes acciones:

Opción 1: se debe solicitar primero la cantidad de números a introducir y a continuación leer todos los números introducidos (almacenándolos en un *array*).

Opción 2: se debe ordenar de forma descendente la secuencia de números (ordenar el *array*) y mostrar por pantalla dicha ordenación.

Opción 3: se debe solicitar primero un número a buscar, mostrándose por pantalla un mensaje que indique si dicho número está o no en la secuencia introducida y otro mensaje indicando el método de búsqueda utilizado.

Opción 4: se debe finalizar la ejecución del programa.

<u>Notas</u>:

- Recuerda modularizar tu programa de la forma que creas más conveniente
- Declara una constante numérica en tu programa que establezca la cantidad máxima de números a introducir (por ejemplo, un máximo de 10 números)
- Puedes utilizar cualquier método de ordenación de tablas de los vistos en clase de teoría
- Implementa los métodos de búsqueda lineal y binaria, y utiliza el más eficiente cuando se elija la opción 3 del menú.

<u>Ejemplo de ejecución</u>:

// Se elige la opción 1…

Introduce la cantidad de números a introducir: 6

Introduce un número: 27

Introduce un número: 16

Introduce un número: 123

Introduce un número: 5
Introduce un número: 2003
Introduce un número: 98

// Se elige la opción 3…
Introduce el número que buscas: 5
El número buscado SI está en la lista
Se ha utilizado el método de búsqueda lineal

// Se elige la opción 2…
La lista de números ordenados de forma descendente es:
2003 ; 123 ; 98 ; 27 ; 16 ; 5

// Se elige la opción 3…
Introduce el número que buscas: 15
El número buscado NO está en la lista
Se ha utilizado el método de búsqueda binaria

// Se elige la opción 3…
Introduce el número que buscas: 16
El número buscado SI está en la lista
Se ha utilizado el método de búsqueda binaria

// Se elige la opción 4…
FIN DE LA EJECUCIÓN DEL PROGRAMA

Propuesto 3. **Implementar el juego "Conecta 4" en C. En primer lugar se pedirá que el usuario introduzca las dimensiones del tablero, que deberán ser siempre iguales o mayores que 4 y menores que 10. Se mostrará entonces el tablero con sus posiciones vacías, y bajo cada columna su índice (las posiciones vacías se representarán mediante el carácter '.'):**

```
Numero de filas: 4
Numero de columnas: 5
.....
.....
.....
.....
.....
12345
```

El jugador deberá introducir el índice de la columna en la que introducir la ficha, y esta ficha caerá a la posición más baja no ocupada de dicha columna. Las fichas del primer jugador se representarán mediante el carácter 'O'.

```
(J1) Introduzca columna (1-5): 3
.....
```

```
.....
.....
.....
..O..
12345
```

Tras esto, se solicitará que el segundo jugador introduzca su jugada. Sus fichas se representarán mediante el carácter 'X'. Tras cada jugada se comprobará si la ficha introducida forma una fila de 4 en la horizontal, en la vertical, o en alguna de sus diagonales.

```
(J2) Introduzca columna (1-5): 3
.....
.....
.....
..X..
..O..
12345
```

El juego terminará cuando alguno de los jugadores logre formar una fila de 4, o bien el tablero esté lleno.

```
.....
....X
...XO
..XXO
.XOOO
12345
El jugador 2 gana la partida.
```

Ayuda: Recuerda modularizar correctamente el programa. Por ejemplo, se pueden definir módulos para comprobar si hay 4 fichas del mismo jugador alineadas en el entorno de un punto **(i,j)** del tablero. Necesitaremos módulos para comprobar la horizontal, la vertical, y las dos diagonales que cruzan el punto: **compruebaHorizontal**, **compruebaVertical**, **compruebaDiagonalIzq**, **compruebaDiagonalDer**. Estos módulos recibirán como entrada las coordenadas **(i,j)** del punto que queramos comprobar, y nos dirán si a su alrededor hay 4 fichas consecutivas alineadas del mismo tipo que la ficha central. Crea además todos los módulos que crear convenientes para implementar las funcionalidades del juego.

Propuesto 4. **Implementar en C el juego del ahorcado. En el programa guardaremos una lista de posibles palabras (con todas sus letras en mayúsculas). Al comenzar la partida se seleccionará una palabra aleatoriamente, y se mostrará la palabra enmascarada con caracteres de subrayado ('_'). Por ejemplo, si la palabra seleccionada es "PROGRAMA" aparecerán 8 caracteres de subrayado:**

```
_ _ _ _ _ _ _ _
```

Tras esto se solicitará al usuario que introduzca una letra que piense que pertenece a la palabra. Si la letra pertenece, ésta quedará desenmascarada:

```
Quedan 10 intentos. Introduzca una letra: R
_ R _ _ R _ _ _
```

El juego terminará cuando la palabra completa quede desenmascarada, o cuando se agote el número máximo de intentos.

Ayuda:

- Recuerda modularizar tu programa de la forma que creas más conveniente.
- El tamaño máximo de las cadenas será de **20** caracteres, tendremos **5** posibles palabras a seleccionar en la lista, y el número máximo de intentos será **10**.
- Podemos crear una lista de cadenas con todas las posibles palabras que puedan aparecer en el juego de la siguiente forma:

```
TPalabra palabras[NUM_PALABRAS] = { "PROGRAMA", "LENGUAJE", "OBJETIVO", "PRACTICA",
                                    "NUMERO" };
```

- Para seleccionar una palabra de la lista de forma aleatoria al iniciar el juego puedes generar números aleatorios tal como se vio en la práctica 5.
- Puedes crear una variable de tipo cadena que almacene la cadena enmascarada actual (se inicializará con tantos caracteres de subrayado como caracteres tenga la palabra seleccionada).
- Cuando el usuario introduzca un carácter deberemos comprobar que sea alfanumérico, y de no ser así pediremos un nuevo carácter.
- Si el carácter introducido está en minúsculas, deberemos transformarlo a mayúsculas.
- Puedes utilizar las siguientes funciones de C++ para el manejo de caracteres:

  bool isalpha(char)

  devuelve true si el carácter es una letra minúscula (‘a’..’z’) o mayúscula (‘A’..’Z’)

  devuelve false en otro caso

  bool islower(char)

  devuelve true si el carácter es una letra minúscula (‘a’..’z’)

  devuelve false en otro caso

  char toupper(char)

  devuelve en mayúscula el carácter pasado como parámetro
- Se comprobarán las coincidencias del carácter introducido dentro de la palabra seleccionada, y se escribirá este carácter en las posiciones correspondientes de la cadena enmascarada.
- Tras cada jugada se puede comprobar con strcmp() si la palabra enmascarada coincide con la original. Si fuese así, el jugador habría ganado la partida.

Ejemplo de ejecución:

```
_ _ _ _ _ _ _ _
Quedan 10 intentos. Introduzca una letra: R
_ R _ _ R _ _ _
Quedan 10 intentos. Introduzca una letra: E
_ R _ _ R _ _ _
Quedan 9 intentos. Introduzca una letra: a
_ R _ _ R A _ A
Quedan 9 intentos. Introduzca una letra: 2
Error: Debe introducir una letra
Quedan 9 intentos. Introduzca una letra: p
P R _ _ R A _ A
Quedan 9 intentos. Introduzca una letra: O
```

```
P R O _ R A _ A
Quedan 9 intentos. Introduzca una letra: m
P R O _ R A M A
Quedan 9 intentos. Introduzca una letra: G
P R O G R A M A
¡Enhorabuena! Ha acertado la palabra
```

Propuesto 5. **Implementar en C un programa que resuelva el error del ejercicio resuelto 6 que se produce cuando en los parámetros un elemento está repetido 3, 4, 5… veces. La solución propuesta cuenta como 2, 3, 4,… repeticiones diferentes. Si un elemento está repetido varias veces, sólo debería contar como 1 en el cómputo de elementos repetidos.**

ERRORES DE PROGRAMACIÓN COMUNES

- **Olvidar que el primer índice en lenguaje C de un array es 0.** En muchas ocasiones se comete el error de empezar por 1.
- **Utilizar como índice un valor que no sea entero.** Los índices que se utilizan para acceder deben ser de tipo entero.
- **Acceder a una posición del array fuera de rango.** Cuando se accede a los elementos de un array no se hace ninguna comprobación. Suele ser un error frecuente acceder a posiciones que no pertenecen al array. Hay que asegurarse que siempre utiliza como índice un valor comprendido entre 0 y *elementos-1*, siendo *elementos* el número de elementos del array.
- **Acceder a un array multidimensional de forma inadecuada.** Cuando se quiere, por ejemplo, acceder a la componente i, j de un array M, no utilice la expresión M[i,j]. En su lugar, utilice la expresión M[i][j].
- **Olvidar contabilizar el carácter de fin de cadena de caracteres.** Cuando reserve espacio para una cadena de caracteres hay que tener en cuenta que la cadena debe incluir el final el carácter nulo.
- **Usar comillas simples para representar cadenas de caracteres.** Las comillas simples se utilizan para representar caracteres individuales. Para las cadenas de caracteres se utilizan dobles comillas.
- **No cerrar bien una cadena de caracteres.** Asegúrese de que las cadenas de caracteres utilizadas como valores constantes se encierran entre dobles comillas.
- **Copiar una cadena en otra que no tiene espacio suficiente.** Cuando copie una cadena en otra, asegúrese de que la cadena destino tiene espacio suficiente para almacenar la cadena que se copia.